Research Institute for Advanced Computer Science
NASA Ames Research Center

Efficient Packing of Patterns in Sparse Distributed Memory by Selective Weighting of Input Bits

Pentti Kanerva

RIACS Technical Report 91.08

March 1991

Efficient Packing of Patterns in Sparse Distributed Memory by Selective Weighting of Input Bits

Pentti Kanerva

Research Institute for Advanced Computer Science
NASA Ames Research Center, M/S Ellis
Moffett Field, CA 94035
E-mail: kanerva@riacs.edu

RIACS Technical Report 91.06

March 1991

Abstract. When a set of patterns is stored in a distributed memory, any given storage location participates in the storage of many patterns. From the perspective of any one stored pattern, the other patterns act as noise, and such noise limits the memory's storage capacity. The more similar the retrieval cues for two patterns are, the more the patterns interfere with each other in memory, and the harder it is to separate them on retrieval. This paper describes a method of weighting the retrieval cues to reduce such interference and thus to improve the separability of patterns that have similar cues.

The Research Institute for Advanced Computer Science is operated by the Universities Space Research Association, The American City Building, Suite 311, Columbia, MD 21044, (301) 730-2656.

The work reported herein was supported in part by Cooperative Agreement NC2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA), and in part by the Academy of Finland.

Efficient Packing of Patterns in Sparse Distributed Memory by Selective Weighting of Input Bits*

Pentti KANERVA

1. INTRODUCTION

The mathematical theory of sparse distributed memory [3] was developed assuming that both the addresses of the storage locations, or hard addresses, and the stored patterns and their retrieval cues are a uniform random sample of all possible n-bit patterns. In the terminology of artificial neural nets, assuming a uniform random distribution of the hard addresses is equivalent to saying that the input coefficients of the hidden layer of a fully connected feed-forward net are set randomly to −1s and 1s with equal probability. This choice of coefficients works well when the data fill the space more or less uniformly. Unfortunately, natural data, such as the bit patterns derived from spoken words, never do. Instead, they cluster, and much of the space remains unoccupied. The result is that the parts of the memory where the clusters fall are overutilized, with stored patterns interfering with each other heavily, while the rest of the memory is underutilized.

One approach to utilizing the memory more efficiently is to fit the hard addresses to the data. An extreme case of this is the Hamming network, which has one storage location for each item of the data set, with the retrieval cue for that item as its address, and the item is stored in that location only. In distributed memories, the hard addresses have been chosen in various ways: Joglekar [2] has used patterns from the data set itself as hard addresses in experiments with NETtalk data, Danforth [1] has used patterns from speech at large as hard addresses in experiments with spoken-digit recognition, Rogers [4] has used genetic algorithms to arrive at a set of hard addresses in experiments with weather data, and Saarinen *et al.* [5] have used Kohonen's self-organization algorithm to arrive at a set of hard addresses in experiments with raster pictures of digits. The back-propagation of error, to modify the input coefficients of the hidden layer, accomplishes a similar thing, the difference being that the back-propagation architectures typically have few hidden units, whereas sparse distributed memories have many.

2. THE PROBLEM

A complementary approach is discussed in this paper: Assuming that the hard addresses of a sparse distributed memory are fixed and nonoptimal for the data, how

* To appear in O. Simula (ed.), *Proceedings of the International Conference on Artificial Neural Networks (ICANN-91, Espoo, Finland)* (Amsterdam: Elsevier).

to address the memory so that patterns associated with similar cues—and stored patterns at large—interfere with each other as little as possible?

The purpose of addressing a sparse distributed memory is to select a small set of storage locations, referred to as the *active locations*, in which an item is stored or from which it is retrieved. The active locations for an item are selected based on the distance between the item's retrieval cue and the hard addresses: The locations closest to the cue are activated.

The standard algorithm for addressing uses Hamming distance, which means that all bits are weighted equally. Consequently, the sets activated by two very similar cues have a large overlap, and when one of them is used to retrieve its associated pattern, the pattern associated with the other is mixed with it heavily (in proportion to the size of the overlap) and makes the output ambiguous.

The standard addressing algorithm is modified by modifying the computation of the distance between a retrieval cue and a hard address. Instead of weighting all bits equally, each bit of each cue is weighted individually, and the locations with the smallest *weighted* distance to the cue are activated. The problem then is to find a vector of nonnegative weights for each item in the data set—for each retrieval cue—such that the set of locations activated by any one cue overlaps as little as possible with the sets activated by the others. This is the sense in which we try to fit a given set of data as well as possible into a given set of storage locations.

Expressed in symbols (italic for *scalars*, bold lowercase for ***vectors***, bold uppercase for ***MATRIXES***), the problem is the following: We are given a data set of t pairs of binary vectors $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, $i = 1, 2, \ldots, t$, where the $\boldsymbol{x}_i$ are t unique n-bit retrieval cues (the input patterns) and the $\boldsymbol{y}_i$ are their associated r-bit patterns to be stored (the desired output patterns). We are also given an $m \times r$ matrix $\boldsymbol{A}$ of bits, interpreted as m hard addresses of a sparse distributed memory (m locations with n-bit addresses; m hidden units with n inputs each).

The set of active locations for the input $\boldsymbol{x}_i$ is determined by computing the Hamming distances between $\boldsymbol{x}_i$ and the rows of $\boldsymbol{A}$ and by selecting the k closest rows, where k is a parameter of the model (k is of order square root of m; in the original sparse distributed memory model, distance below a threshold is used as the activation criterion). Because there can be several rows at the maximum activation distance, we in fact select all the rows (and locations) that are no further than the k closest and indicate their number by k^+ (the exact value of k^+ depends on $\boldsymbol{x}_i$ and $\boldsymbol{A}$ [and on $\boldsymbol{w}_i$, see below]).

For convenience, and without loss of generality, we let the two values of a binary variable (a bit) be −1 and 1 when dealing with addresses and with retrieval cues. Choosing the k^+ closest rows according to Hamming distance is then equivalent to choosing the k^+ rows with the largest inner product with $\boldsymbol{x}_i$, so that the k^+ largest com-

ponents of the vector Ax_i (= c_i) indicate the active locations for the input x_i. Designate this set by a_i (a_i is an m-bit activation vector, with k^+ ones indicating the k^+ active locations and zeros elsewhere; $a_{iv} = 1$ iff $c_{iv} \geq$ the kth-largest component of c_i).

Weighting the bits of a retrieval cue individually can now be expressed as follows. The weights for x_i are an n-dimensional vector of nonnegative real numbers. Call it w_i, and let W_i be the corresponding diagonal matrix. Then the activation vector a_i will indicate the k^+ largest components of AW_ix_i instead of Ax_i. We can now state the problem as finding t weight vectors w_i such that the corresponding t activation vectors a_i cover the space of all possible activation vectors—that is, vectors of k^+ ones and $n - k^+$ zeros—as uniformly as possible.

This statement does not specify the problem fully, because we are not saying how to maximize uniformity. Two possibilities come readily to mind, but there can be others. One is by minimizing the average overlap and the other by minimizing the maximum overlap over all pairs of activation vectors for the data set, where the overlap between two activation vectors is the number of ones in their logical AND (which is the number of hard locations activated by both of two input patterns). The two methods described in this paper are heuristic and do not necessarily achieve either of these minimizations, but they do improve the utilization of the memory by spreading the total activity rather uniformly over all the storage locations.

Finally, how are the active locations used? The memory stores r-bit patterns in an $m \times r$ matrix C, which is initially set to zeros. A row of A and the corresponding row of C constitute a storage location (a hard location), with the row of A as its address and the row of C as its contents. The pattern y_i is stored by adding it to the contents of each active location, and a pattern is retrieved by adding the contents of the active locations and by thresholding the resulting r sums. In symbols, storing (x_i, y_i) means adding the matrix $a_i y_i^T$ to C, and retrieving the pattern for x_i means thresholding the vector $C^T a_i$, where a_i is the activation vector for x_i and ^{T} means transpose (these are the standard outer-product or Hebbian learning rule and the corresponding output rule, respectively).

3. PROPERTIES OF THE WEIGHT VECTOR FOR A RETRIEVAL CUE

In the following, we use the terms *weighted distance* and *new distance* from x_i to x_j to mean the Hamming distance weighted with w_i: $d_i(x_j) = w_i^T\langle x_j \neq x_i\rangle$ ($\langle a \neq b\rangle$ is the vector with 1s where a and b differ, and 0s elsewhere). To make the weighted distances comparable with each other and with the Hamming distance, we normalize the weights so that they add to n, $\sum_v w_{iv} = n$ ($w_{iv} \geq 0$, $v = 1, 2, \ldots, n$). Notice that the weighted distance from x_i to x_j is usually different from the weighted distance from x_j to x_i, and also that the activation vectors are not affected by the normalizing of the

weights.

How are the weights for x_i affected by the other retrieval cues in the data set? If x_j is far from x_i (at about $n/2$ bits or further), their activation vectors based on uniform weighting are well separated, so that x_j does not have a great influence on the weights for x_i. We merely need to make sure that the new distance to x_j not be too small. If x_j is near x_i, their initial activation vectors—those based on uniform weighting—have a large overlap, and the weights need to be chosen to decrease it. Therefore, the new distance to x_j needs to be much larger than the old. The new distance is influenced only by the weights for the coordinates at which x_j differs from x_i, and it equals the sum of the weights for those coordinates. The larger this sum is, the larger is the new distance, and the smaller is the overlap between the new activation vectors. From these observations we conclude that x_j should increase the weights for x_i at the places where it differs from x_i, and that the smaller the number of places at which it differs—that is, the closer the two initially are—the larger should the increase be.

4. WEIGHTS BASED ON THE RETRIEVAL CUES

If the distribution of the hard addresses is approximately uniform, a good set of weights can be derived from the retrieval cues alone. The heuristic algorithm that I have used for calculating the weights is as follows: Each x_j contributes to w_i an incremental weight vector $u_j = \langle x_j \neq x_i \rangle / h(x_i, x_j)$, where $h(..)$ is the Hamming distance; and w_i is the (normalized) sum of the incremental vectors over all $j \neq i$. Using the reciprocal of the Hamming distance in the increments is based on a simple probabilistic argument, and experiments showed that the reciprocal was better than either its square or square root.

5. WEIGHTS BASED ON THE OUTPUT SUMS

A more general method of weighting is gotten by looking at the memory's output. First, we store all t retrieval cues—or, rather, the corresponding vectors of 0s and 1s—"autoassociatively" in the standard manner, meaning that the data are of the form (x_i, y_i), where $y_{iv} = 0$ if $x_{iv} = -1$, and 1 otherwise, and that the unweighted distance is used in computing the activation vectors. To determine the weights w_i, we then do a standard memory retrieval with x_i as the input, but instead of taking the final thresholded output, we work with the n sums s_{iv} ($s_i = C^T a_i$). What can they tell us?

Since every point of the space of n-bit vectors is equivalent to every other point, we can simplify the discussion by assuming that $x_i = -\langle 11...1 \rangle$ (the vector of -1s) and y_i is the zero-vector (XOR all addresses and data with y_i, and replace s_{iv} by $N - s_{iv}$ if $y_{iv} = 1$, where N is the total number of patterns stored in the locations activated by x_i). The places where x_j differs from x_i will them be the 1s of y_j, and they occur in the sum vec-

tor (*i.e.*, y_j occurs in it) multiplied by the size of the overlap of the activation vectors a_i and a_j. As discussed above, this is what we want, so that the sum vector s_i should provide a good set of weights for x_i when $x_i = -\langle 11...1 \rangle$; when $x_i \neq -\langle 11...1 \rangle$, the sum vector is first transformed as shown above. It also needs to be transformed further.

I performed a number of experiments to determine how to transform the sums into the input weights and arrived at the function

$$w'_{iv} = N'^{(s_{iv}/N')^E},$$

where w'_{iv} is the weight before normalization, N' is the total number of patterns other than y_i that have been stored in the locations activated by x_i (it is the sum of the sizes of the overlaps of activation vector a_i with all the other activation vectors a_j; $N' = N - k^+$), and $E \geq 0$ is a parameter (E is the same for all patterns in any given experiment). Thus, the unnormalized weights for x_i are between 1 and N'. Pleasing about this solution to the weights is that it implies the distribution of the hard addresses.

6. EXPERIMENTS

Data. I experimented with three sets of data. In the first set a certain bit is particularly significant in discriminating between patterns of the set, the second set has a tight cluster of patterns in one part of the space (four highly significant bits), and the third set is "natural" bit patterns for the capital letters.

The first data set has 32 31-bit patterns ($t = 32$, $n = 31$) arranged symmetrically so that the distances from any one pattern to the others are 1, 2, 3, ..., 31 bits. The 31 bits are divided into five fields of 1, 2, 4, 8, and 16 bits, and each field is set to all 0s or all 1s according to the bits of the binary numbers from 0 to 31. The first five patterns thus are (0)(00)(0000)(00000000)(0000000000000000), (1)(00)(0000)(8 + 16 0s), (0)(11)-(0000)(24 0s), (1)(11)(0000)(24 0s), and (0)(00)(1111)(24 0s) (the parens show grouping into fields). In this data set, each bit is 0 or 1 with equal probability, but the first bit is more important than any other in discriminating between patterns, and bits 2 and 3 are very important, whereas none of the last 16 bits is particularly important.

The second data set has 28 36-bit patterns, which are organized in fields of length 1, 1, 1, 1, 4, 4, 12, and 12 bits. The first 16 patterns consist of all 16 binary combinations in bits 1–4, with the rest of the patterns being 0s. In patterns 17–22, bits 1–4 act as a 4-bit field. These patterns, in hex, are 0F0000000, FF0000000, 00F000000, F0F000000, 0FF000000, and FFF000000. In patterns 23–28, bits 1–12 act as a 12-bit field. These last six patterns are 000FFF000, FFFFFF000, 000000FFF, FFF000FFF, 000FFFFFF, and FFFFFFFFF. In this data set, the first four bits are highly significant: They define a tight cluster of 16 patterns about the origin. Bits 5–12 are also signifi-

cant and, together with bits 1–4, define a loose cluster about the origin. The mean distance between the patterns of this data set is 11.3 bits.

The third data set has 26 35-bit patterns from 7×5 raster images of the capital letters, given here in hex (the 1st 35 bits each of): 2114AFC62, F463E8C7C, 74610845C, F46318C7C, FC21E843E, FC21E8420, 74610BC5E, 8C63F8C62, 71084211C, 388421498, 8CA98A4A2, 84210843E, 8EEB58C62, 8E6B39C62, 746318C5C, F463E8420, 74631ACDE, F463EA4A2, 7460E0C5C, F90842108, 8C6318C5C, 8C62A5108, 8C635AED4, 8C5445462, 8C5442108, and F8444443E. The closest pair of patterns in this set (D and O) is separated by two bits, and five patterns are three bits away and seven are four bits away from the closest other pattern. The mean distance between patterns is 14.1 bits.

Procedure. The different methods of weighting the retrieval cues were compared in a series of experiments. An experiment consisted of four passes. In each pass, the data set was written once into the memory (the contents C were first set to 0s). The addressing of the memory for the first three passes was as follows: (1) uniform weighting of the retrieval cues; (2) using weights based on the output sums of the first pass (see Sec. 5); and (3) using weights based on the retrieval cues (see Sec. 4). The fourth pass was for control: (4) a uniform random data set, of the same size as in the first three passes, was stored with uniform weighting.

All experiments were made with uniform random sparse distributed memories with 300 locations (the address matrix A was set randomly to −1s and 1s with equal probability; $m = 300$). In Pass 1, the minimum number of locations activated by a retrieval cue, k, was 15; in the other passes, k was adjusted so as to give an average k^+ as close to that for the first pass as possible, so that the memory would hold nearly equal numbers of patterns in the different passes. Since the formula for the weights based on the output sums contains a parameter, E, Pass 2 required a number of iterations for finding the optimal exponent E (optimal was defined as the value that minimizes the mean [root mean square] overlap over all pairs of activation vectors for the data set). Each experiment was run ten times on each of the three data sets.

Results. The results are summarized in Tables 1–3. Entries of the form $x(y)z$ give the smallest, the mean, and the largest of the ten values obtained. In the headers, t is the size of the data set, n is the pattern length, *mean* k^+ is the average number of locations activated by a retrieval cue, and E is the exponent used in the weight equation in Pass 2.

The row labeled *Empty cells* gives the number of hard locations that were activated by none of the retrieval cues—they are wasted. *Max. pats. / cell* is the number of times that the "busiest" location was activated and written into. Very busy locations become noise and thus also are wasted. *Var. pats. / cell* is the variance of the number of activations per location (*i.e.*, the number of patterns stored per location). A small variance

TABLE 1

Experiment with Data Set 1 (t = 32, n = 31, mean k^+ = 20.1 (**21.5**) 22.7, E = 1.9 (**2.08**) 2.4)

Pass	1 Uniform weights	2 Weights from sums	3 Weights from data	4 Random data
Empty cells	112 (**129**) 142	74 (**86.3**) 97	71 (**84.1**) 93	18 (**23.9**) 31
Max. pats./cell	12 (**14.1**) 16	7 (**8.0**) 9	8 (**8.3**) 10	6 (**7.7**) 10
Var. pats./cell	7.3 (**8.15**) 9.3	3.23 (**3.84**) 4.53	3.47 (**3.87**) 4.32	1.85 (**2.08**) 2.39
Mean % overlap	25.0 (**26.5**) 29.1	14.7 (**15.8**) 16.6	14.9 (**15.7**) 16.4	10.1 (**10.4**) 11.0
Max. % overlap	100 (**100**) 100	50 (**55.2**) 59	45 (**54.4**) 59	36 (**45.3**) 59

TABLE 2

Experiment with Data Set 2 (t = 28, n = 36, mean k^+ = 19.1 (**21.2**) 23.3, E = 0.20 (**0.25**) 0.32)

Pass	Uniform weights	Weights from sums	Weights from data	Random data
Empty cells	120 (**144**) 159	0 (**6.1**) 11	0 (**1.5**) 5	31 (**38.6**) 49
Max. pats./cell	19 (**21.2**) 22	5 (**5.9**) 7	5 (**6.7**) 9	6 (**6.7**) 8
Var. pats./cell	13.0 (**15.1**) 16.8	1.25 (**1.48**) 1.77	1.16 (**1.54**) 1.81	1.65 (**2.02**) 2.48
Mean % overlap	36.9 (**42.1**) 46.3	10.0 (**10.9**) 11.6	9.64 (**11.2**) 12.1	9.73 (**10.6**) 11.8
Max. % overlap	89 (**98.3**) 100	43 (**56.6**) 74	42 (**53.4**) 65	36 (**44.5**) 65

TABLE 3

Experiment with Data Set 3 (t = 26, n = 35, mean k^+ = 20.9 (**21.8**) 23.3, E = 0.26 (**0.37**) 0.50)

Pass	Uniform weights	Weights from sums	Weights from data	Random data
Empty cells	89 (**103**) 111	27 (**35.7**) 43	33 (**41.4**) 52	29 (**40.9**) 46
Max. pats./cell	10 (**13.5**) 17	5 (**5.6**) 7	5 (**5.8**) 7	5 (**6.7**) 11
Var. pats./cell	4.29 (**5.24**) 6.08	1.31 (**1.47**) 1.60	1.38 (**1.62**) 1.86	1.54 (**1.78**) 2.32
Mean % overlap	18.6 (**20.7**) 22.8	9.4 (**9.94**) 10.4	10.5 (**11,0**) 11.7	9.69 (**10.4**) 11.2
Max. % overlap	73 (**80.6**) 93	32 (**37.9**) 48	38 (**47.5**) 67	32 (**42.1**) 63

means uniform utilization of memory (writing into the locations at random results in a variance equal to the mean). *Mean % overlap* is the average (root mean square) overlap over all pairs of activation vectors for the data set, and *Max. % overlap* is the largest of them. Small percentages mean good discrimination by the memory.

I did these same experiments without clearing the memory between the first and the second pass—before storing the data with the weighted cues. After the data were thus stored twice, unweighted retrieval produced nearly the same set of weights for the weighted retrieval as it did with the data stored only unweighted.

7. CONCLUSIONS

Storing a set of input patterns autoassociatively in a sparse distributed memory provides a means of finding a set of input weights for these same patterns: the sums obtained by reading from the memory can be converted into the weights. Such weighting of the input patterns improves the utilization of the memory if the set of patterns does not match well the set of memory locations. No assumptions are made concerning the distribution of the memory locations, so that this method can be used in addition to any other method, such as redistributing the memory locations, to improve memory utilization. The memory can be used at once for both the unweighted and the weighted storing of the patterns, which points a way to a two-step method of storing a set of patterns in and retrieving it from the memory. The weighting is then like an attentional mechanism: the weights from the first pass indicate where the discriminating features of the pattern are in relation to this particular set of data and this particular set of memory locations.

REFERENCES

[1] Danforth, D. G. (1990) An empirical investigation of sparse distributed memory using discrete speech recognition. *Proceedings of International Neural Network Conference (INNC-90 Paris) 1*:183–186 (Dordrecht, The Netherlands: Kluwer Academic Publishers).

[2] Joglekar, U. D. (1989) Learning to read aloud: A neural network approach using sparse distributed memory. Report No. 89.27, RIACS, NASA Ames Research Center, Moffett Field, California.

[3] Kanerva, P. (1988) *Sparse Distributed Memory.* Cambridge, Massachusetts: MIT Press.

[4] Rogers, D. (1990) Predicting weather using a genetic memory: A combination of Kanerva's sparse distributed memory and Holland's genetic algorithms. In D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2* (San Mateo, Calif.: Kaufmann).

[5] Saarinen, J., Pohja, S., and Kaski, K. Self-organization with Kanerva's sparse distributed memory. In O. Simula (ed.), *Proceedings of the International Conference on Artificial Neural Networks (ICANN-91, Espoo, Finland)* (Amsterdam: Elsevier, in press).